Reconstruction: A Counter-Intuitive Approach to Multi-Layer Design Editing with AI Agents

David Ferris
South Park Commons
research@dferris.me

September 2025 (Work in Progress)

Abstract

Reliable multi-layer editing remains outside the capabilities of current AI systems. We discuss why LLM-only approaches produce broken designs, argue that rendering is equivalent to verification, and explain the failure modes that arise in simple VLLM agent architectures. We also propose a pragmatic new algorithm that leverages modern image editing models to reconstruct and edit layered designs and introduce a new dataset and edit benchmark for multi-layer design editing.

1 Introduction and Related Work

1.1 The Current State of Design AI

Multi-layer editing is the core mode of professional graphic editing in tools like *Figma*, *Photoshop*, and *Canva*: independent layers stack ("composite") to form a final image. Contemporary "text-to-prototype" systems (e.g., Lovable, v0, Bolt) are not true design agents; they rely on HTML flow layouts (e.g., flexbox) that succinctly describe *relative* layouts, where elements are ordered in a hierarchy or graph.

The benefit of flow layouts is robustness to content changes: resizing one element typically re-flows others so text does not overlap, icons stay aligned, and distribution and centering are preserved — i.e., the layout does not "break." If taste and judgment are the pinnacle of good design, broken layouts are the epitome of bad design: universally undesirable mistakes. Much of today's design AI emits HTML precisely to avoid breakage. This difference between absolute and relative positioning is one of the primary challenges in true design AI.

Layout-first approaches like LayoutDM[4] and LayoutGAN[5] suggest generating the layout first, then populating the design. While this can work, design is circularly dependent: the layout affects the content which affects the layout, etc. For this reason we are interested in approaches which generate the layout and content simultaneously.

1.2 The Challenges in Evaluating Design

> "Learn the rules like a pro, so you can break them like an artist." — Pablo Picasso

Evaluating design is challenging because the gap between great and terrible can be narrow. Some decisions are merely poor (e.g. clashing colors); others are *objectively broken* (e.g., text wrapping mid-word). Human experts also disagree on success criteria, making evaluation difficult. Fortunately, the bar for AI design is still low: the near-term aim is to move from broken layouts to passable (if conservative) designs.

Preprint.

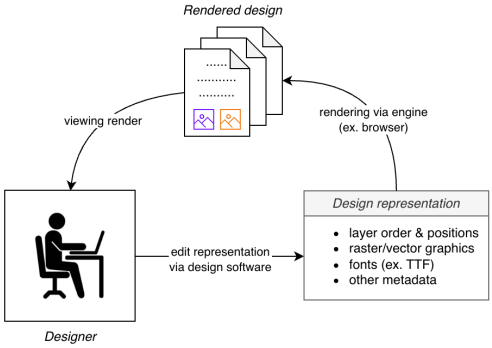

Figure 1: The design-render-edit loop

There are numerous aesthetic evaluation models like AesExpert [3], however, these are trained mainly on photographs and perform poorly when evaluating graphic or user-interface (UI) design quality. UIClip[6] is an open-source model trained on websites by applying jitter functions to break their layouts, however even this model is domain-specific and does not generalize well to evaluating posters, advertisements, or other graphic designs.

1.3 Limitations of LLM-Only Design Agents

> "The map is not the territory." — Alfred Korzybski

LLMs cannot see. Whether the design is represented as absolutely positioned HTML/JSON/XML, *rendering is equivalent to verification*. Asking an LLM to edit layout without rendering is like asking a front-end engineer to write HTML/CSS blind: it is nearly impossible for anything but the simplest designs. This is in large part because of **input sensitivity**: small changes can create large visual regressions. For example, a copy change may reflow text, causing it to overlap with an icon. Flow layouts mitigate this, explaining their prevalence.

Even state-of-the-art reasoning models (e.g., olympiad-capable LLMs) cannot reliably perform simple design tasks because they lack visual feedback. As we show, rendering is therefore essential to verify non-breaking edits.

This rendering requirement reflects real-world design workflows: (1) the designer edits an existing design, (2) the modification is recorded in the underlying representation, and (3) the design is re-rendered for immediate visual feedback. The process repeats iteratively until completion.

1.4 An Ontology of Broken Layouts

By "broken layout", we refer to any design choice that is universally objectionable and clearly an error. For every "rule" in graphic design, it is easy to provide a counter-example; we grant this to be true, however we maintain there are still common errors that can be classified.

2 Methodology

We compare four different approaches to multi-layer AI design editing.

1. **Zero-shot, design-only**: Only the design template is included in context.
2. **Zero-shot, with render**: Both the design spec and render are included in context.
3. **Design Agent**: A 2-agent VQA Loop (Figure 3).
4. **Reconstruction**: The algorithm explained in Section 3.

We also include scores for just the edited image, since this poses an upper bound on reconstruction.

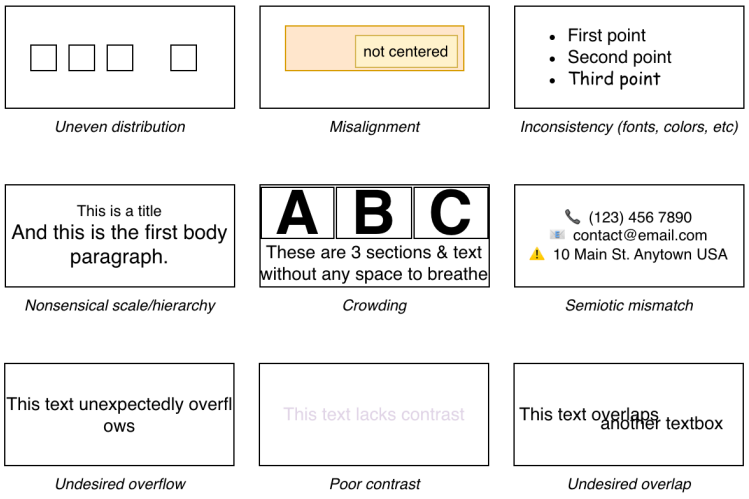

Figure 2: Possible reasons for broken layouts

Failure Mode	**Strategy to Remedy**
Uneven distribution	Elements should be evenly spaced using consistent margins or grid systems to maintain visual balance.
Misalignment	Components should align along shared axes or baselines to reinforce structural coherence.
Inconsistency (fonts, colors, etc.)	Apply a unified visual system—consistent typography, color palette, and iconography—to ensure semantic and aesthetic coherence.
Nonsensical scale/hierarchy	Visual hierarchy should reflect semantic importance through proportional size, weight, and spacing relationships.
Crowding	Adequate whitespace, padding, and line spacing should be preserved to promote legibility and visual breathing room.
Semiotic mismatch	Icons and visual symbols should be selected according to their conventional or intuitive meanings within the given context.
Undesired overflow	Text boxes and containers should dynamically adjust or constrain content to prevent clipping and maintain readability.
Poor contrast	Ensure sufficient luminance and chromatic contrast between text and background for accessibility and legibility.
Undesired overlap	Elements should be positioned to avoid spatial collisions, preserving intended visual grouping and reading order.

Table 1: Common layout failure modes and corresponding mitigation strategies. The list summarizes universal, non-stylistic failure categories that frequently occur in automatically generated designs.

2.1 Data

We selected 100 Free template images from the Canva.com template library, spanning the *Poster*, *Instagram Post*, *Presentation*, and *Website* categories. These designs were chosen to represent a diverse mix of content types, color palettes, visual complexity, and aspect ratios. These template images (.jpeg, .png or .webp) were then reconstructed into multi-layer templates by hand. Only templates explicitly marked as "Free" were used to ensure open availability and legal use in research settings.

Each template was exported as a flattened raster image which, with the help of a manual GUI, was converted into a structured multi-layer representation (via manual recreation of layers). The raster

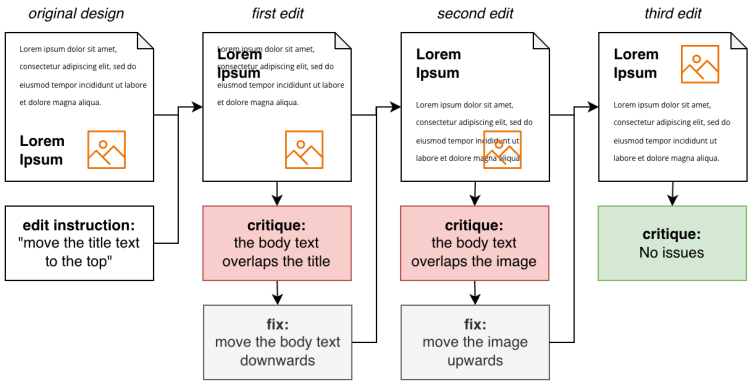

Figure 3: Designer-Critic agent loop. The designer applies edits to a design, and the critic provides feedback about a design and whether an edit was successfully applied.

images served as inputs for reconstruction and evaluation, while the layered representations provided ground-truth structure for quantitative comparison.

To systematically evaluate design editing capabilities, we developed a taxonomy of 15 edit types spanning color, layout, text, image, and style modifications at both global and local scales. These include: (1) Color Palette Variation and Accent Recolor for chromatic adjustments; (2) 2-Element Layout Swap, Global Layout Change, Object Removal/Addition, and Hierarchy Adjustment for spatial modifications; (3) Translation, Copy Expansion, Typography Style Shift, and Add Secondary Text Block for textual transformations; (4) Image Replacement and Background Replacement for visual content changes; and (5) Contrast & Readability Enhancement and Theme Restyling for stylistic refinements.

For each of the 100 designs, we used `GPT-4o` to generate design-specific edit instructions conditioned on the rendered design image and each edit type's meta-prompt template, yielding 1,500 unique instructions (100 designs $\times$ 15 types). The model was instructed to produce concise, imperative commands under 30 words that are contextually appropriate while conforming to each edit type's constraints, representing reasonably comprehensive coverage of realistic design edits.

All figures and examples in this paper are derived from Canva Free templates or their synthetic variants.[1]

2.2 Evaluation

2.2.1 Aesthetics

We use a frontier VLLM, `gemini-2.5-pro`, to evaluate design quality. The model is prompted to act as a human-like graphic design critic, producing a holistic score from 0–100 based on clarity, hierarchy, typography, alignment, contrast, and overall craft. The full text of the evaluation prompt is provided in Appendix A.

To assess the reliability of this automated critic, we conducted a qualitative sanity check using intentionally regressed versions of high-quality designs (e.g., with degraded alignment, color balance, or typography). As shown in Figure 9 and Appendix 8, the critic consistently assigns lower scores to these degraded variants, aligning with human visual intuition. While not a substitute for a formal user study, this comparison provides initial evidence that the approach captures meaningful aspects of design quality.

[1]Templates were used under the Canva Content License Agreement. `https://www.canva.com/policies/content-license-agreement/`.

2.2.2 Edit Quality

To measure edit success, we use a similar approach to that proposed in DPG-Bench [2] and break the edit into specific sub-tasks and use VQA to evaluate each one independently. `gemini-2.5-pro` is also used for this VQA evaluation.

3 Reconstruction: Leverage Image Editing Models for Multi-Layer Edits

Image editing models (e.g., GPT-Image, Nano-Banana, Flux.1 Kontext) operate in pixel-space: they accept multi-modal inputs and render outputs. This makes them incompatible with multi-layer editing, since they are only capable of modifying a single, flattened image. It is initially unclear how to best leverage such a model to perform structured, multi-layer edits.

However, there is one critical property of modern image editing models: *they almost never produce broken layouts*. Trained on vast datasets of visually coherent images, their generative process inherently favors plausible arrangements of elements, proper alignment, and aesthetic harmony. This hints at a powerful, if counter-intuitive, workaround: instead of using these models for direct layer manipulation, we can use them for holistic, pixel-space editing and then reconstruct a layered structure from their output.

We propose a multi-stage process that leverages the strengths of both layered representations and pixel-space generative models. The workflow, illustrated in the figure below, consists of three main steps: compose, edit, and decompose.

1. **Compose (Render):** The process begins with an existing multi-layer design, represented in a structured format (e.g., JSON, SVG). This design is rendered into a single, flat bitmap image. This step effectively erases the explicit layer information, creating the input required by the image editing model.
2. **Pixel-Space Edit:** The rendered image is passed to a generative image editing model along with a high-level natural language prompt (e.g., "make this look more professional," "change the style to a vintage 70s aesthetic"). The model, operating entirely in pixel-space, generates a new image that reflects the requested changes while maintaining visual coherence. This is the core of the stylistic transformation, where the model's implicit understanding of design principles is applied holistically.
3. **Decompose (Reconstruct):** The final, and most critical, step is to reverse the initial rendering process. The newly generated flat image is analyzed to reconstruct a multi-layer design. This decomposition pipeline involves several sub-tasks:
   - **Element Segmentation and Detection:** Computer vision models are used to identify and isolate distinct elements. This includes Optical Character Recognition (OCR) to locate and transcribe text, object detection to find icons and logos, and semantic segmentation to identify background shapes and image blocks.
   - **Property Extraction:** For each detected element, we extract its visual properties. For a text element, this includes its content, bounding box, color, and an estimation of its font family, weight, and size. For shapes and images, we extract their geometry, color fills, or image crops.
   - **Layer Reassembly:** The detected elements and their extracted properties are reassembled into a new, structured, multi-layer representation. The z-ordering of layers is inferred from occlusion in the generated image.

This reconstructive approach allows us to treat the powerful but unstructured image model as a "black-box" design consultant. We leverage its aesthetic capabilities to generate a high-quality visual target, then use more traditional computer vision techniques to recover the editable, multi-layer structure that is essential for professional design workflows. It is a pragmatic bridge, enabling multi-layer editing by repurposing models that were never designed for it.

3.1 Detecting Layers

After the image editing model produces a flattened output, we segment it into individual text elements using Gemini 2.5 Flash for OCR (system prompts in Appendix B.1 and B.2). The model returns structured JSON with each text element's content, bounding box coordinates, and color. After

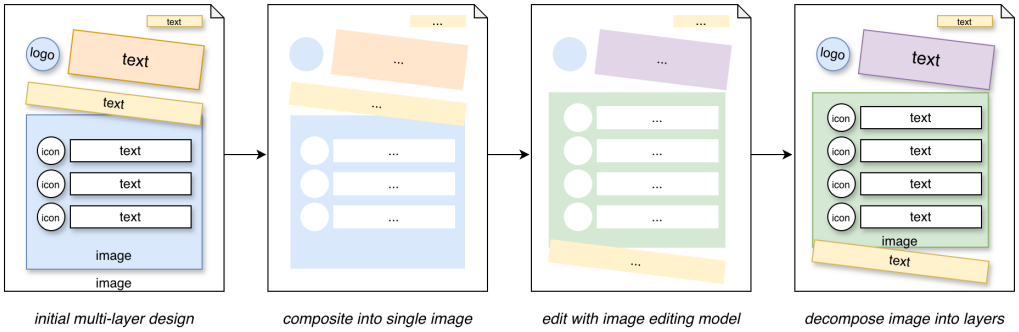

Figure 4: The reconstruction algorithm

detection, we crop each text region and pass it to the ResNet-50 classifier to extract font properties (family, size, weight, etc).

To recover a clean background layer, we use Gemini 2.5 Flash Image Preview for two-pass inpainting (Appendix B.3).

3.2 Extracting Text Properties

Accurate text property extraction is critical for reconstruction: once text regions are identified via OCR, we must determine their visual styling—font family, size, weight, color, and alignment—to reassemble an editable layer. While OCR provides the textual content, extracting these typographic properties requires a separate classification step.

We approach font detection as a supervised classification problem. Given a cropped text region, we predict the font family from a constrained vocabulary of approximately 50 common typefaces. This constraint reflects the practical reality of professional design: most designs rely on a small set of widely-available fonts (Google Fonts, Adobe Fonts, system defaults) rather than the full universe of all possible typefaces.

3.2.1 Synthetic Data Generation

We generate a large-scale synthetic training corpus using Playwright and headless Chromium to programmatically render text across font families. Our pipeline samples 10,000 diverse phrases from a curated list spanning casual speech, technical jargon, and common design copy to ensure the model generalizes across varied textual content. We select 180 popular font families representative of common design choices: 150 Google Fonts, 6 system fonts, and 24 other free online fonts.

For each phrase-font pair, we introduce realistic visual variation to improve model robustness. Font sizes are uniformly sampled from 10–100px, with independent random padding on all four sides (0–150px). Text alignment is randomly chosen from left, center, or right. To ensure the model handles diverse color schemes, 50% of samples use randomized background and text colors with sufficient luminance contrast ($\geq$4.5:1 ratio, approximating WCAG AA standards), while the remaining 50% use black-on-white. Font weight is evenly split between normal and bold.

Each configuration is rendered in Chromium and captured as a PNG image via the Playwright screenshot API. This process yields 500 training images per font, resulting in 90,000 total samples (examples in Appendix E). An 85/15 train-validation split produces 76,500 training samples and 13,500 validation samples.

During training, the data loader applies additional augmentations including random resized crops (scale 0.8–1.0), small affine transformations ($\pm 2^\circ$ rotation, $\pm 2\%$ translation, $\pm 2^\circ$ shear), photometric jitter ($\pm 10\%$ brightness and contrast), and Gaussian blur (15% probability).

3.2.2 Model Architecture and Training

We fine-tune a ResNet-50 [1] convolutional network initialized with ImageNet-1K pre-trained weights. The final fully-connected layer is replaced with a dropout layer (p=0.2) followed by a linear classifier projecting to 13 font classes.

Training proceeds in two stages:

1. **Warmup (3 epochs)**: The ResNet backbone is frozen, and only the classification head is trained. Batch normalization layers remain in evaluation mode to preserve ImageNet-trained statistics. This prevents catastrophic forgetting and stabilizes early training.
2. **Full fine-tuning (up to 50 epochs)**: All layers are unfrozen. We employ discriminative learning rates: the backbone receives a smaller learning rate (1e-4) while the head receives a larger rate (3e-4). Training uses AdamW optimization with weight decay (1e-4), cosine annealing, gradient clipping (max norm = 1.0), label smoothing (0.1), and early stopping with patience of 6 epochs.

All images are converted to grayscale (then replicated to 3 channels for compatibility with ImageNet weights), resized to 384×384 with center cropping, and normalized using ImageNet statistics. Data augmentation during training includes random crops, affine transformations, color jitter, and Gaussian blur.

3.2.3 Font Detection Results

The model achieves 71.2% top-1 accuracy on the hold-out validation set with a median per-class accuracy of 74.3%. Performance varies significantly across font categories: highly distinctive decorative fonts like Oi (97.8%), Ashing (95.5%), and Audiowide (94.5%) achieve near-perfect classification, while similar font variants prove challenging—most notably Roboto Condensed (1.3%), which is heavily confused with its parent family Roboto (12.4%). Other condensed geometric sans-serifs like League Gothic (7.4%) and handwriting fonts such as Patrick Hand (12.7%) also struggle. This aligns with human perception: distinguishing between closely related font variants (e.g., a condensed version versus its regular counterpart) or subtle geometric sans-serif differences is inherently difficult even for trained designers. The full confusion matrix can be found in Appendix F.

4 Results

All agents regressed the quality of the design when making edits to varying degrees. Predictably, both design and edit quality improved with agent complexity: more tokens allocated to visual reasoning and critique returned greater edit and design performance.

Table 2: Agent Performance: Design Quality vs. Edit Quality

Agent	Original Design	Design Quality	Edit Quality
Original	79.1 ± 1.0	–	–
SingleShot	–	67.3 ± 1.1	61.8 ± 1.9
MultiShot	–	72.0 ± 0.9	69.4 ± 1.7
CriticEditor	–	70.8 ± 1.0	73.1 ± 1.8
ImageEdit	–	75.6 ± 1.2	71.2 ± 1.6

Figure 5: Agent(s) performance summary (TO-DO OLD DATA REPLACE THIS)

Interestingly, direct image editing (ImageEdit) is not the clear editing front-runner. Breaking down edit quality by category reveals its spiky performance, relative to the VLM agents. It excels at stylistic, color, and image editing tasks but it's performance on text editing drags the average downwards.

Table 3: Design and Edit Quality by Category.

	Design Quality					Edit Quality				
Method	**Color**	**Image**	**Layout**	**Style**	**Text**	**Color**	**Image**	**Layout**	**Style**	**Text**
Original	78.3	80.7	79.6	76.2	78.8	–	–	–	–	–
SingleShot	70.4	69.7	75.1	72.9	71.5	68.6	60.2	55.9	60.3	77.4
MultiShot	68.1	68.9	73.2	70.8	70.1	78.5	60.6	80.4	62.7	88.3
ImageEdit	74.9	78.2	77.7	76.6	76.4	82.3	92.8	70.5	82.1	60.9

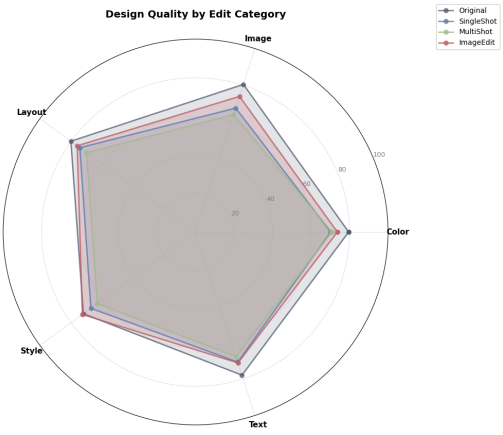

Figure 6: Design accuracy by category.

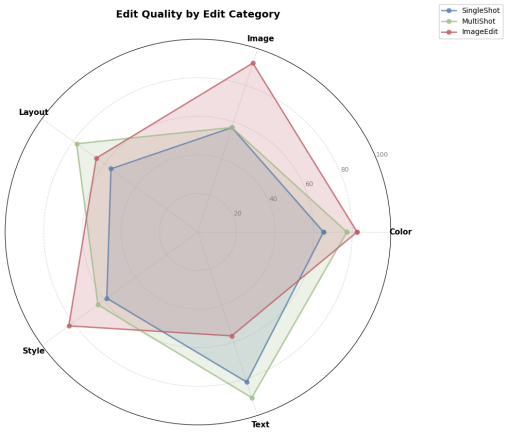

Figure 7: Edit accuracy by category.

5 Limitations & Future Work

Reconstruction is a powerful rethinking of the multi-layer design editing problem. However, there are several limitations in the current approach that remain open areas for improvement. While image editing models can extract raster layers, we do not yet propose a method to reliably recover vector

graphics. The text property detection model could also be extended to support a wider variety of fonts, weights, and rich text effects like curved text.

Evaluation can likewise be improved: although current off-the-shelf VLMs are reliable and approximate human-level aesthetic preferences, they still underperform skilled human designers. An aesthetic model trained specifically to evaluate graphic design would enable more effective hill-climbing techniques.

Another area for future work is in maintaining design consistency. As presented, the reconstruction algorithm makes no guarantees about brand (font, text size, colors, etc) consistency *between* edits. It seems plausible that these initial design criteria could be used as suggestions for deconstruction.

This research also demonstrates a glaring need for better design evaluation methods. VLM-based evaluation, while reasonably accurate, is imprecise, frequently failing to detect broken layouts. Better evaluation would enable better hill-climbing on the design agent problem. A proper design critic should not only have excellent visual reasoning against design principles, but be generally aligned with skilled human designer preferences.

6 Conclusion

Reconstruction is, at its core, a strategic inversion of the design-editing problem. Rather than forcing models to reason symbolically about layouts they cannot see, we rely on what they are good at: producing visually coherent images. By treating pixel-space editing as the target and recovering structure afterward, we bypass the brittleness that plagues LLM-only agents and obtain designs that are both coherent and editable. The approach is not without limitations: vector fidelity, fine-grained typography, and cross-edit consistency all remain open problems, but it already offers a surprisingly effective path to high-quality multi-layer editing. In the longer run, reconstruction highlights an important design principle: the future of design AI will likely require models that fluidly couple symbolic representations with visual reasoning, rather than expecting either modality to substitute for the other.

References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[2] Xiwei Hu, Rui Wang, Yixiao Fang, Bin Fu, Pei Cheng, and Gang Yu. Ella: Equip diffusion models with llm for enhanced semantic alignment, 2024.

[3] Yipo Huang, Xiangfei Sheng, Zhichao Yang, Quan Yuan, Zhichao Duan, Pengfei Chen, Leida Li, Weisi Lin, and Guangming Shi. Aesexpert: Towards multi-modality foundation model for image aesthetics perception. *arXiv:2404.09624*, 2024.

[4] Naoto Inoue, Kotaro Kikuchi, Edgar Simo-Serra, Mayu Otani, and Kota Yamaguchi. Layoutdm: Discrete diffusion model for controllable layout generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10167–10176, 2023.

[5] Jianan Li, Jimei Yang, Aaron Hertzmann, Jianming Zhang, and Tingfa Xu. Layoutgan: Generating graphic layouts with wireframe discriminators. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019.

[6] Junxiao Wu, Yu-Hsuan Peng, Aoyu Li, Amanda Swearngin, Jeffrey P. Bigham, and Jeffrey Nichols. Uiclip: A data-driven model for assessing user interface design. In *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology (UIST '24)*, New York, NY, USA, 2024. Association for Computing Machinery.

A Full Critic Evaluation Prompt

```
You are a meticulous human-like graphic design critic. Given a single image
```

```
of a designed layout (poster, social tile, ad, slide, etc.), silently evaluate
overall quality as a person would-holistically, not by rules alone-then output
**one integer from 0-100**. Do not explain your reasoning or output any words,
units, symbols, or punctuation-**only the number**.

Evaluate using the rubric below (weights sum to 100). Consider context-agnostic
usability for a general audience unless clear intent is visible. Reward clarity,
craft, and effectiveness; penalize obvious defects. After judging, combine
criteria into a single normalized score and round to the nearest integer, clamped
to [0,100].

1) Purpose & message clarity (15) - Is the primary idea instantly understandable?
   Is there a clear focal point and sensible call to action or takeaway?
2) Visual hierarchy & information architecture (12) - Logical ordering, scannability,
   sensible grouping, scale used to rank importance.
3) Typography & legibility (12) - Type pairing, sizing, line length/leading, tracking/
   kerning, case, readability across background, no orphan/widow issues.
4) Alignment & grid discipline (8) - Columns/baseline/grid coherence; elements
   truly centered when intended; edges line up.
5) Spacing & breathing room (8) - Adequate margins, padding, and gutters;
   no crowding; comfortable negative space.
6) Consistency of styles (6) - Fonts, weights, colors, icon styles, corner radii,
   stroke widths; coherent system use.
7) Contrast & accessibility (8) - Foreground/background contrast (aim $\geq$ WCAG-ish
   body-level contrast), color used to separate layers and states.
8) Color harmony & tone (6) - Palette fits message; saturation and temperature
   balanced; no jarring clashes unless clearly intentional.
9) Imagery/illustration quality & relevance (6) - Resolution, cropping, lighting,
   subject relevance; no artifacts or watermarks.
10) Iconography & semantics (6) - Icons match meaning; pictograms unambiguous;
    no semiotic mismatch.
11) Balance, rhythm & flow (6) - Visual weight distribution, compositional balance
    (rule-of-thirds/axis), eye path through the layout.
12) Craft/technical execution (5) - No compression artifacts, banding, jaggies;
    crisp edges; exports sized appropriately.
13) Originality, brand/tone fit & polish (2) - Feels professional and intentional;
    style fits an inferred brand or purpose.

Explicitly check and penalize when present (do not limit evaluation to these):
uneven distribution, misalignment, elements ‘‘not centered’’ when claimed,
inconsistency of fonts/colors/styles, nonsensical scale/hierarchy, crowding/no
breathing room, semiotic/icon mismatch, undesired text/image overflow or cropping,
poor contrast, undesired overlaps, illegible small text, sloppy shadows/glows,
low-res or mismatched icons, awkward rag, broken grids, excessive effects,
color clashes, irrelevant stock imagery.

Scoring instructions:
- Judge holistically first (overall human impression), then adjust with the rubric.
- Map the final impression to 0-100 (0 = unusable/chaotic; 50 = serviceable but
  clearly flawed; 75 = good with minor issues; 90 = excellent; 100 = exemplary,
  production-ready).
- Round to nearest integer and **output only that integer**.
  If the image is blank/unreadable, output 0.
```

B Gemini OCR System Prompts

B.1 Initial Step

```
Output a json list where each entry contains the 2D bounding box
in "box_2d", the text content in "label", and the text color in
"color". The bounding box coordinates should be normalized to the
image dimensions (0-1000). Here's what the response should look
like: { "box_2d": [68, 75, 322, 408], "label": "Start working out
now", "color": "#0055b3" }
```

B.2 Secondary Consolidation

```
I have detected the following text boxes in an image. Each has an
index, a label, color, and bounding box coordinates ([yMin, xMin,
yMax, xMax]). Some of these are single lines that belong to a
larger, multi-line text block. Please group the indices of text
boxes that should be merged.

IMPORTANT RULES:
- Only group text boxes that have the EXACT SAME COLOR
- Text boxes that are far apart or have different styling should not
be grouped
- Different colors indicate different text elements and must remain
separate

Return a JSON object with a single key "groups". [...] For example:
{ "groups": [[0, 1], [3, 4, 5]] }. Do not include single, ungrouped
boxes in the response.
```

B.3 Background Extraction

```
IMPORTANT: Remove ONLY the following specific text elements from the
image: [list].

CRITICAL PRESERVATION RULES:
- DO NOT remove any images, photos, illustrations, or visual elements
- DO NOT remove logos, icons, graphics, or decorative elements
- DO NOT remove borders, frames, backgrounds, or visual design
elements
- DO NOT remove colors, patterns, textures, or visual styling
- ONLY remove the exact text content specified above
- Preserve all non-text visual elements completely unchanged
- Fill any removed text areas naturally to match surrounding
background/context

Return only the edited image with text removed but all other visual
elements preserved.
```

C Aesthetic Scoring Examples

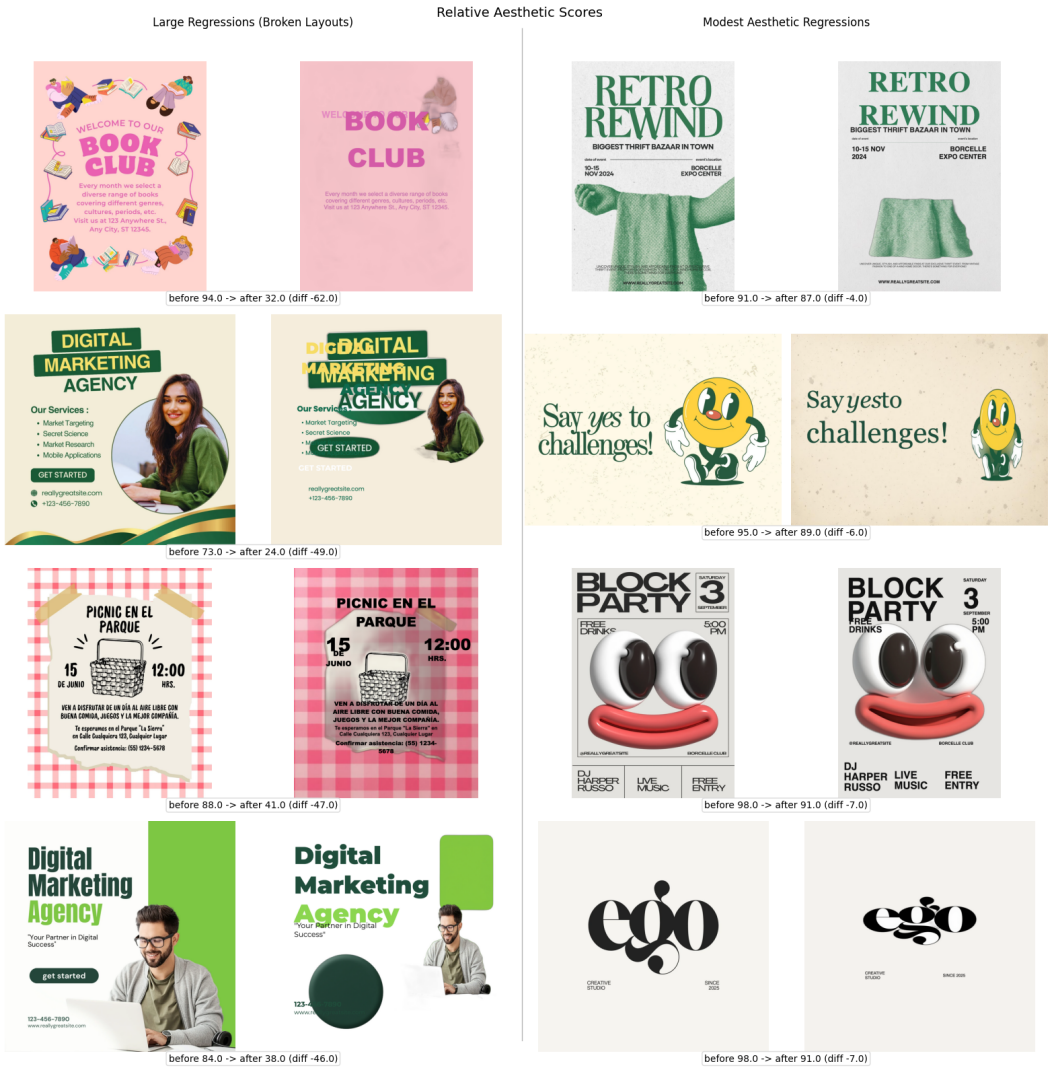

Figure 8: Examples of large aesthetic regressions (left) and modest ones (right)

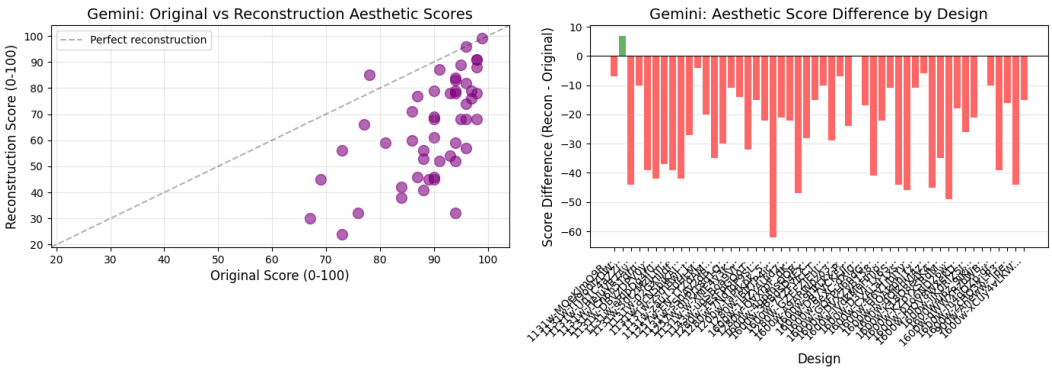

Figure 9: Obvious regressions are correctly identified almost all of the time

D Layout Regression Example

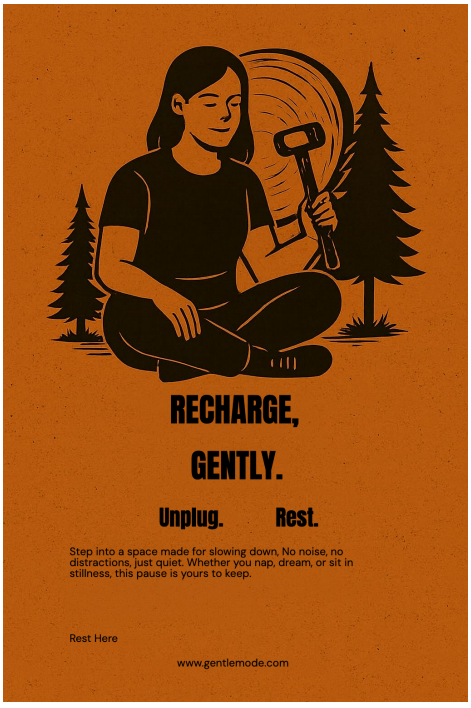

Figure 10: Original poster.

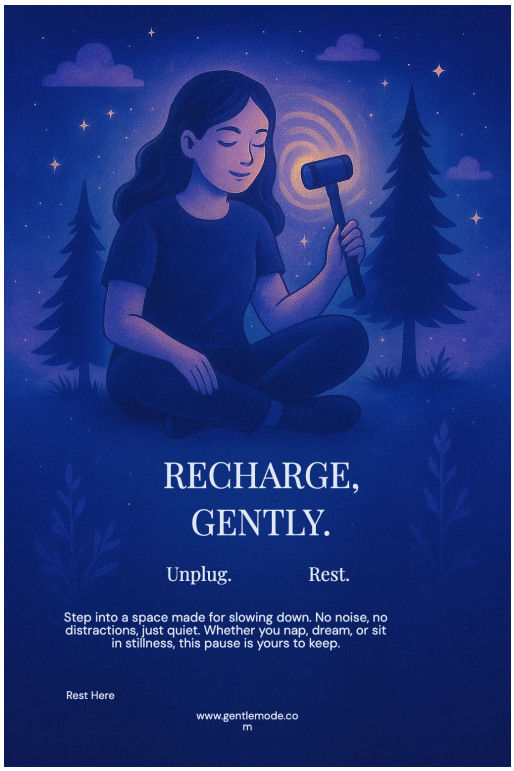

Figure 11: Edited vibe ("dream-like"): sensible colors/fonts, but alignment/text wrapping breaks.

E Font Dataset Example

Figure 12: Examples from the synthetic font classification dataset

F Font Detection Confusion Matrix

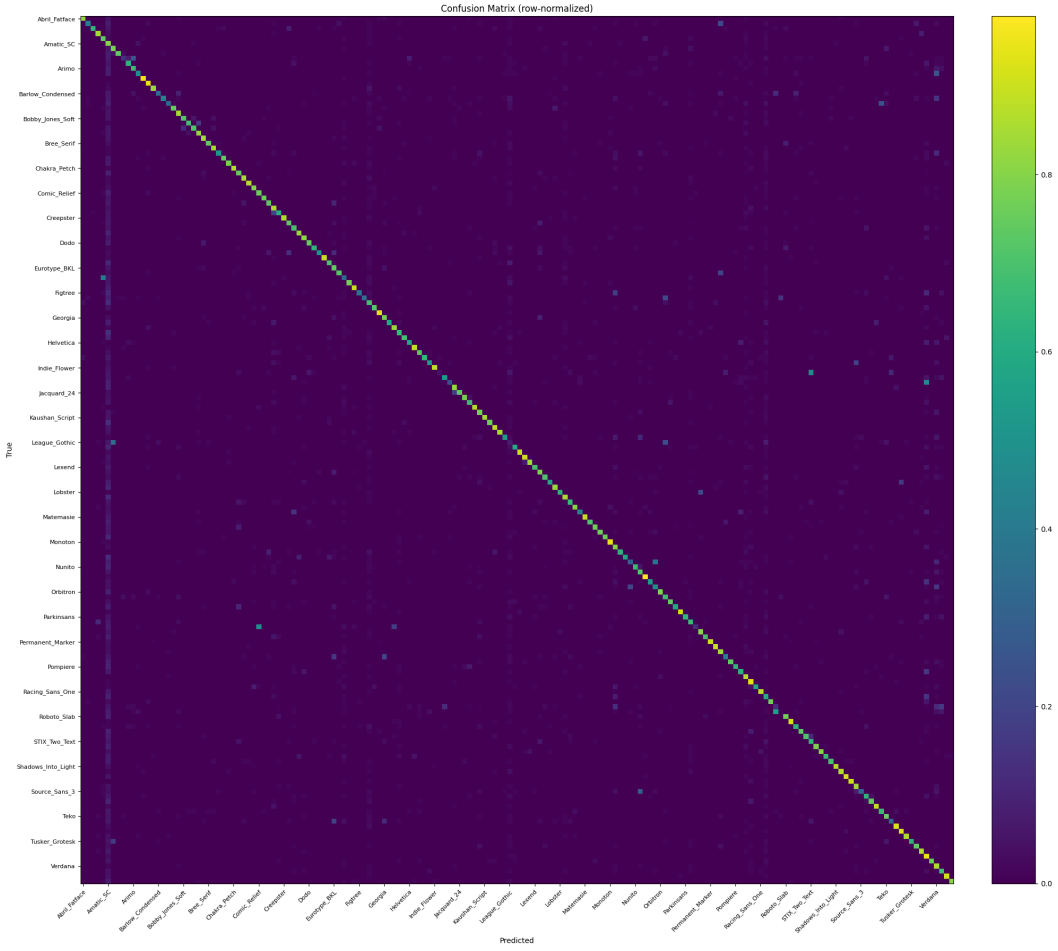

Figure 13: Confusion matrix for 180 font classes